Transpiling Applications into Optimized Serverless Orchestrations

Short Paper

Joel Scheuner

scheuner@chalmers.se

@joe4dev

Philipp Leitner

Supported by
What is Serverless Computing?

Event → Triggers → Function → Store

Database
(service, HTTP, …)
Serverless Application

Composition Problem
Functions into apps

“I want to sequence functions”
“I want to run functions in parallel”
“I want to select functions based on data”
“I want to retry functions”
“I want try/catch/finally”
“I have code that runs for hours”

Source: AWS re:Invent 2016: NEW LAUNCH! Serverless Apps with AWS Step Functions (SVR201)
Composition Problem

“We need better orchestration for serverless workflows to make system design more straightforward and easier to implement”
Lessons learned experimenting with an AWS Lambda orchestration engine, blog 2017 by Ben Kehoe

“I’m looking for better ways to compose and re-use functions and serverless resources, cloudformation just doesn’t cut it”
My wish list for AWS Lambda in 2018, blog 2018 by Yan Cui

“We don’t yet have the Rails of serverless—something that doesn’t necessarily expose that it’s actually a Lambda function under the hood.”
Serverless is eating the stack and people are freaking out—as they should be, blog 2018 by Forrest Brazeal

"composition and testing of functions […] sparsely covered by current scientific literature but […] immensely important in practice"
A mixed-method empirical study of Function-as-a-Service software development in industrial practice, JSS 2019

“serverless frameworks need to provide a way for tasks to coordinate”
Cloud Programming Simplified: A Berkeley View on Serverless Computing, technical report 2019

“Research will need to focus on what composition models would fit FaaS, on ways to express these compositions of functions, and on how to support (frequent) function-updates and hybrid-cloud deployment.”
The SPEC Cloud Group’s Research Vision on FaaS and Serverless Architectures, WOSC 2017
The Serverless Trilemma (ST)

ST-safe iff:

1. Functions considered as black boxes

2. Compositions of functions should be functions themselves

3. No double billing
Composition Approaches

Function Fusion

Function Coordinator

Function Workflows

Function Chaining

Event-Driven Function Composition

by database or by queue

Background:
Function Composition in a Serverless World, Kubeconf 2018
Serverless Apps with AWS Step Functions, AWS re:invent 2016
Function Orchestration Systems

AWS Step Functions

Azure Durable Functions

Apache OpenWhisk Composer

Comparison of Production Serverless Function Orchestration Systems, 4th WoSC 2018
Function Orchestration Systems

AWS Step Functions

Azure Durable Functions

Apache OpenWhisk Composer

Function-focus ➔ Application-focus

Function-focus

Application-focus
Composition Vision

*source-to-source transformation of the abstract syntax tree (AST)*
Prototype Implementation

Transpile

Feedback

Monitor

Javascript

Application Code

Serverless Orchestration

IBM Cloud Functions

Facebook jscodegsift with recast

Apache OpenWhisk Composer

JS

Apache OpenWhisk is a serverless event-based programming service and an Apache Incubator project. https://openwhisk.apache.org/

Composer is a new programming model for composite

Facebook / jscodegsift

A JavaScript codegen toolkit.

338 commits

1 branch

2019-06-16

Chalmers | University of Gothenburg
Transpilation Example

```javascript
function f1() {
    return { message: 'f1' };
}

function f2() {
    return { message: 'f2' };
}

function f3() {
    return { message: 'f3' };
}

f1();
f2();
f3();

composer.sequence(
    composer.action('f1', { action: f1 }),
    composer.action('f2', { action: f2 }),
    composer.action('f3', { action: f3 })
);

var value = 6;
if (value % 2 === 0) {
    console.log(value / 2);
}

composer.let({
    value: 6
}, composer.if(params => value % 2 === 0, params => console.log(value / 2)));
```
Composition Example

Visual Recognition Application

Verify URL -> valid -> Download -> Classify -> Format -> Filter

Log Error

```json
[{
  "className": "giant panda, panda, panda bear, [...]",
  "probability": 0.99935364723228577
},
{
  "className": "American Staffordshire terrier, [...]",
  "probability": 0.00012968324881512672
},
{
  "className": "Arctic fox, white fox, Alopex lagopus",
  "probability": 0.00008463481208309531
}]

[{
  "className": "tiger cat",
  "probability": 0.4486467832873535
},
{
  "className": "lynx, catamount",
  "probability": 0.4224271774291992
},
{
  "className": "tabby, tabby cat",
  "probability": 0.07799174636602402
}]

[{
  "className": "giant panda, panda, panda bear, [...]",
  "probability": 0.99935364723228577
},
{
  "className": "American Staffordshire terrier, [...]",
  "probability": 0.00012968324881512672
},
{
  "className": "Arctic fox, white fox, Alopex lagopus",
  "probability": 0.00008463481208309531
}]

[{
  "className": "tiger cat",
  "probability": 0.4486467832873535
},
{
  "className": "lynx, catamount",
  "probability": 0.4224271774291992
},
{
  "className": "tabby, tabby cat",
  "probability": 0.07799174636602402
}]
```
Example Transformation

```javascript
if(verifyUrl(url)) {
    var img = download(url);
    var prediction = classify(img);
    var label = format(prediction);
    return filter(result);
} else {
    return logError();
}
```

composer.if(composer.action('verifyUrl', { action: verifyUrl }),
            composer.sequence(
                composer.action('download', { action: download }),
                composer.action('classify', { action: {
                    kind: 'blackbox',
                    image: 'jamesthomas/action-nodejs-v8:tfjs',
                    code: 'const main = ${classify}',
                    memory: 512 } }),
                composer.action('format', { action: format }),
                composer.action('filter', { action: filter })
            ),
            composer.action('logError', { action: logError }));
```
Composition Performance (1)

Execution Time* [ms]

<table>
<thead>
<tr>
<th></th>
<th>Verify URL</th>
<th>Download</th>
<th>Classify</th>
<th>Format</th>
<th>Filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cold</td>
<td>300</td>
<td>1200</td>
<td>1300</td>
<td>300</td>
<td>300</td>
</tr>
<tr>
<td>Warm</td>
<td>2</td>
<td>600</td>
<td>700</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

*exemplary measurements
Composition Performance (2)

Execution Time* [ms]

Cold  1400  1300  (warm) 4
Warm  600   700   4

*exemplary measurements
Composition Cost

Monthly costs* [USD]

<table>
<thead>
<tr>
<th>Memory (MB)</th>
<th>Cost [USD]</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>1.3</td>
</tr>
<tr>
<td>256</td>
<td>2.6</td>
</tr>
<tr>
<td>512</td>
<td>5.2</td>
</tr>
</tbody>
</table>

*based on 1’000’000 warm-start requests per month

**Pricing based on: [https://cloud.ibm.com/openwhisk/learn/pricing](https://cloud.ibm.com/openwhisk/learn/pricing)
$0.000017 per second of execution, per GB of memory allocated
Benefits

• More accessible to build serverless applications
  • Transpilation from generic JS to platform-specific code
• Faster application runtime
  • Automated function fusion
• Cheaper computation cost
  • Targeted function size
Limitations

• Function fusion only when code available → violating ST black-box constraint
• Harder to debug at runtime
• Data marshalling overhead and limitations
• Integration into third party services
Future Work

• Extend transpilation prototype
  • Support more composition primitives

• Integrate and evaluate dynamic deployment alternatives
Conclusion

Thursday 9:00 – 10:30 in N440:

**Tutorial 5: Performance Benchmarking of Infrastructure-as-a-Service (IaaS) Clouds with Cloud WorkBench**
Discussion Input
How should serverless compositions be expressed?

As data?

```json
```

As code?

```javascript
final StateMachine stateMachine = stateMachine()
    .comment("A demo Sequence state machine")
    .startAt("f1")
    .state("f1", taskState()
        .resource("arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME")
        .transition(next("f2")))
    .state("f2", taskState()
        .resource("arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME")
        .transition(next("f3")))
    .state("f3", taskState()
        .resource("arn:aws:lambda:REGION:ACCOUNT_ID:function:FUNCTION_NAME")
        .transition(end()))
    .build();
```

```javascript
module.exports = composer.sequence(  composer.action('f1'),
  composer.action('f2'),
  composer.action('f3'),
);
```

```javascript
f1();
f2();
f3();
```
Should machines decide upon deployment structure?

• Is it practical (e.g., understandable) to have dynamically changing deployment structures?
  • Debugging (source maps)?
  • Testing?

```
if(verifyUrl(url)) {
  var img = download(url);
  var prediction = classify(img);
  var label = format(prediction);
  result = filter(result);
} else {
  logError();
}
```

```
composer.if(composer.action('verifyUrl', { action: verifyUrl })),
  composer.sequence(
    composer.action('download', { action: download })),
    composer.action('classify', { action: { kind: 'blackbox', image: 'jamesthomas/action-nodejs-v8:tfjs', code: `const main = ${classify}`, memory: 512 } )},
    composer.action('format', { action: format })),
    composer.action('filter', { action: filter })),
    composer.action('logError', { action: logError })),
```

Entering composition[1].consequent[2]"
Which application types benefit from this approach?

• Which applications have heterogenous-enough footprints?
Any related work from (other) communities?

- Programming Languages (PL)
- Domain Specific Languages (DSL)
- Workflows
- …

✉️ scheuner@chalmers.se
Serverless Background

Source: © 2018 IBM Corporation
Serverless Pros and Cons

- **Containers**
  - + Tools
  - + Control and Flexibility
  - + De Facto Standards

- **Functions**
  - + Fine-Grain Metering
  - + Faster Autoscaling
  - + Event-driven Programming

Source: © 2018 IBM Corporation
Serverless Application Types

Serverless is **good** for:
- short-running
- stateless
- event-driven

Serverless is **not good** for:
- long-running
- stateful
- number crunching

- Microservices
- Mobile Backends
- Bots, ML Inferencing
- IoT
- Modest Stream Processing
- Service integration

- Databases
- Deep Learning Training
- Heavy-Duty Stream Analytics
- Numerical Simulation
- Video Streaming

Source: Slides Workshop of Serverless Computing (WoSC'4), 2018
**Abstract Syntax Tree (AST)**

```javascript
var value = 6;
if (value % 2 === 0) {
    console.log(value / 2);
}
```

Tree Visualization using AST Explorer: [https://astexplorer.net/](https://astexplorer.net/)
function transform(file, api, options) {
  imports.register(j, imports.config.CJSBasicRequire);
  const { statement } = j.template;
  const parsed = j(file.source)
parsed.find(j.CallExpression)
  .replaceWith(function (path) {
    const actionName = path.value.callee.name;
    const left = j.memberExpression(
      j.identifier('module'),
      j.identifier('exports'))
    const right = j.callExpression(
      j.memberExpression(
        j.identifier('composer'),
        j.identifier('action'))
    , [
      j.literal(actionName),
      createActionReference(actionName)
    ]
    return j.assignmentExpression(
      '=',
      left,
      right,
    )
  })
  const transformed = parsed.addImport(statement`
    const composer = require('openwhisk-composer');`)
  const outputOptions = {
    quote: 'single'
  }
  return transformed.toSource(outputOptions);
}
AWS Lambda Power Tuning

https://github.com/alexcasalboni/aws-lambda-power-tuning